#1. Write a template function isListSorted. It returns true if the elements in the parameter STL list 'lst' are in the ascending order (determined by <).

ANSWER:

```cpp
template<typename T>
bool isSorted(const list<T> & lst)
{
    if (lst.size() < 2)
    {
        return true;
    }
    else
    {
        list<T>::const_iterator it1 = lst.begin(); // points to first element
        list<T>::const_iterator it2 = lst.begin(); // points to second element
        it2++;

        // As long as the 2nd iterator is in the list
        while (it2 != lst.end())
        {
            if (*it2 < *it1) // not in the sorted order
                return false; // early exit
            else
                // advance both iterators
                it1++; it2++;
        }
        // All adjacent elements are in the sorted order.
        // Return true as the final result.
        return true;
    }
}
```

#2 For the doubly-linked List class you wrote for HW#2, add the following function:
void merge(List& x);

ANSWER:

This function is rather involved, and YOU DON'T HAVE TO WORRY ABOUT IT. However, I recommend you read my solution below and figure out how I'm doing this task.
The logic, in particular the traversal over two sorted Lists for merge, while keeping the existing iterators intact, may require some thinking.

```cpp
void merge(List& x)
{
    if (this != &x)
    {
        iterator p1 = begin();
        iterator p2 = x.begin();
        iterator p3;

        // Strategy is to advance p1 until it points to an element
        // which is larger than the element p2 currently points to.
        // Then we insert the node pointed by p2 into *this just before p1.
        while (p1 != end())
        {
            // Advance p1
            while (p1 != end() && *p1 < *p2)
                p1++;

            // If the loop above terminated because it went off the list,
            if (p1 == end())
                break; // break out of the loop (to go to (***)
            else
                // Insert node pointed by p2 just before p1
                p3 = p1;
                p1 = p2;
                p2 = p3;
        }
```
while (*p2 < *p1) {
    // save p2's next in p3
    p3 = p2.current->next;
    p2.current->prev->next = p2.current->next; // (1)
    p2.current->next->prev = p2.current->prev; // (2)
    p1.current->prev->next = p2.current; // (3)
    p1.current->prev = p2.current; // (4)
    theSize++;
    x.theSize--;
    p2 = p3; // essentially to advance p2
}
// (***)
// If any element is still left in x, we splice all of them
// (in one operation) into/at the end of *this.
if (p2 != x.end()) {
    // Being lazy here -- calling splice().
    // Note at this point, p1 is pointing at this->end(), and
    // x contains the remaining elements (only).
    splice(p1, x);
}

#3. Below is a pseudocode for an algorithm which checks whether or not a given
text (a string) has balanced delimiters {}, (), and []. For example, "[ ( ) ]" is balanced but "{ ( ) }" is not balanced. Write a program, using a STL stack and
file input.
ANSWER: See "balancedApp.cpp" (a separate file)

#4. For each of the following six program fragments: a. Give the Big-Oh complexity.

(1) sum = 0;
    for( i = 1; i <= n; ++i )
        ++sum;
ANSWER: O(n)

(2) sum = 0;
    for( i = 1; i <= n; ++i )
        for( j = 1; j <= n; ++j )
            ++sum;
ANSWER: O(n^2)

(3) sum = 0;
    for( i = 1; i <= n; ++i )
        for( j = 1; j <= n * n; ++j )
            ++sum;
ANSWER: O(n^3)

(4) sum = 0;
    for( i = 1; i <= n; ++i )
        for( j = 1; j <= i; ++j )
            ++sum;
ANSWER: O(n^2)

(5) sum = 0;
    for( i = 1; i <= n; ++i )
        for( j = 1; j <= i * i; ++j )
            ++sum;
for( k = 1; k <= j; ++k )
    ++sum;

ANSWER: O(n^5) -- outer-most i is O(n), then j-loop is O(n^2), and k--loop is O(n^2)

(6) sum = 0;
    for( i = 1; i <= n; ++i )
        for( j = 1; j <= i * i; ++j )
            if( j % i == 0 )
                for( k = 1; k <= j; ++k )
                    ++sum;

ANSWER: Never mind this question.. Too much.
But if you want to know, the answer is O(n^4) -- k--loop only execute when j is
divisible by i, thereby reducing the factor by O(n)

#5. Using STL list, write TWO versions of a procedure which receives two list<int>
objects and returns true if the two lists are equal (i.e., having the same
elements) or false otherwise. Assume neither list contains duplicates.

ANSWER: Here a general approach is described rather than C++ code.

* O(N^2) solution -- Compare each element from the first list with every element
in the second list for an equal element. If no such equal element is found in the
second list, the procedure terminates immediately with false. If every element in
the first list found an equal element in the second list, the procedure returns
true.

* O(NlgN) solution -- First sort both lists, which will take O(NlgN) time for
each list. Then traverse the two lists simultaneously, which takes O(N) time,
comparing elements at the same index. If the elements compared are not the same
(including the case when one list was shorter than the other), the procedure
returns false immediately. Otherwise, if the traversal didn't terminate along the
way, the procedure returns true at the end.

#6. Below is the function BubbleSort.

void Bubblesort(int data[], int n)
{
    int tmp,i,j;

    for (i=0; i<n-1; i++) {
        for (j=0; j<n-i-1; j++)
            if (data[j] > data[j+1]) { // (**)
                tmp = data[j];
                data[j] = data[j+1];
                data[j+1] = tmp;
            }
    }
}

a. What are the worst case and the average case big-Oh complexities of the method?
ANSWER: O(N^2) for both cases.

b. Show the content of an array {5, 3, 2, 4} when the function is called with the
array (and its length 4 as the 2nd parameter). Show the array for EACH i and j
combination.

ANSWER:

initially | 5 | 3 | 2 | 4 |
          +---+---+---+---+

i=0, j=0 | 3 | 5 | 2 | 4 |
          +---+---+---+---+

i=0, j=1 | 3 | 2 | 5 | 4 |
          +---+---+---+---+
Note that the function executes exactly those iterations (and in this order) and no more or less.

#7. [Textbook Exercise 4.46] Two trees $T_1$ and $T_2$ are isomorphic if $T_1$ can be transformed into $T_2$ by swapping left and right children of (some of the) nodes in $T_1$. For instance, the two trees in the figure below are isomorphic because they are the same if the children of A, B and G, but not the other nodes, are swapped. Give a polynomial time algorithm to decide if two trees are isomorphic.

**ANSWER:**

Using the (public) template `BinaryNode` struct:

```cpp
template <typename T>
struct BinaryNode {
    T element;
    BinaryNode *left;
    BinaryNode *right;

    BinaryNode( const T & theElement, BinaryNode *lt, BinaryNode *rt )
        : element( theElement ), left( lt ), right( rt ) { }
};
```

The function can be written as something like this. The parameter pointers t1 and t2 are supposed to point to two trees $T_1$ and $T_2$ respectively, and they 'traverse' the trees in a synchronized manner.

```cpp
// Top-level call
template <typename T>
bool isIsomorphic(const BinaryNode<T> *t1, const BinaryNode<T> *t2) {
    if (t1 == NULL && t2 == NULL) // both reached null at the same time
        return true;
    else if (t1 == NULL || t2 == NULL) // only either one is null
        return false;
    else {
        // logic is (1) the element values of both nodes must be the same,
        // (2) look at all combinations of left/right children of the two nodes,
        // and if any combination succeeds, that's
        return (t1->element == t2->element &&
            (isIsomorphic(t1->left, t2->left) && isIsomorphic(t1->right, t2->
            right))
        ) || (isIsomorphic(t1->left, t2->right) && isIsomorphic(t1->right, t2->
            left)));
    }
}
```

#8. Show the result of inserting 8, 3, 6, 16, 2, 11, 15 into an initially empty binary search tree. Then show the result of deleting the root.

**ANSWER:**
After inserting 8, 3, 6, 16, 2, 11, 15

```
8
/   \
3   16
/   /   \
2 6 11
   \
15
```

After deleting the root

```
11
/   \
3   16
/   /   \
2 6 15
```

#9. Show the result of inserting 2, 1, 6, 3, 4, 5, 7, 9, 8 into an empty AVL tree. Draw the tree after each insertion.

ANSWER:

AVL after inserting 2, 1, 6, 3, 4

```
2
/   \
1 6 case 2 1 4
/ at 6 /   \
3 ======> 3 6
\ 4
```

insert 5

```
2
/   \
1 4 case 4 2 6
/ at 2 /   /   \
3 6 ======> 1 3 5
/ 5
```

insert 7, 9

```
4
/   \
2 6 /   /   \
1 3 5 7
\ 9
```

insert 8

```
4
/   \
2 6 case 3 2 6
/ at 7 /   /   \
1 3 5 7 ======> 1 3 5 8
\ 9
```

#10. Show the result of inserting 10, 22, 31, 4, 15, 28, 17, 88, 59 into an empty Top-down Red-Black tree. Draw the tree after each insertion.

ANSWER:

(1) insert 10 (2) insert 22 (3) insert 31

```
10
10
10
22
```
(4) insert 4

```
22  color flip  22  insert  22
/\  at 22 ==> /\  4R ==> /\
10R 31R 10 31 10 31 4R
```

(5) insert 15  (6) insert 28

```
22
/\ 10 31
/\ 4R 15R 4R 15R 28R
```

(7) insert 17

```
22  color  22
/\ 10 31  flip at 10R 31 17R ==> 10R 31 17R
/\ 10 15R 28R 4 15 28R 4 15 28R 17R
```

(8) insert 88

```
22
/\ 10R 31
/\ 4 15 28R 88R
```

(9) insert 59

```
22  color  22
/\ 10R 31  flip at 10R 31R 9R 10R 31R
/\ 10R 31  31 ==> 10R 31R 9R 10R 31R 31R
/\ 15 28R 88R 4 15 28 88 4 15 28 88 88 17R 17R 17R 59R
```

#11. Priority Queues.

a) For the heap below, show the result of inserting 1.

```
  5
 / \ +---------------------------------------------+
 8 50  5  8  50  11  10  52  55  25  22  20
 / \ / \ +---------------------------------------------+
11 10 52 55
 / \ /  
25 22 20
```

ANSWER:

(1) First place 1 at the end of the queue

```
  5
 / \ +---------------------------------------------+
 0 1 2 3 4 5 6 7 8 9 10
```
(2) Swap 1 and 10 (because 10 > 1)

(3) Swap 1 and 8 (because 8 > 1)

(4) Swap 1 and 5 (because 5 > 1) --the final tree/queue

b) For the heap below, show the result of applying DeleteMin.

ANSWER:

(1) Pop the root (4) and replace it with the last element in the queue (50).

(2) Swap 50 and 11 (because 11 < 17 and 50 > 11)
(3) Swap 50 and 19 (because 19 < 20 and 50 > 19) --the final tree/queue

```
                   11
                   /   \
                  19   17
                 /     /\     \
                22   50 20 60
                /         /\    \
               65         30
```

#12. When a heap of N elements are stored in an array, and the elements are stored from index 0 (as shown in the lecture note, NOT 1 as in the textbook), what is the index of the 'last internal node'?

ANSWER: parent of the last leaf node, found at index (n - 1 - 1) / 2 = (n - 2) / 2

#13. Given an array [3, 1, 8, 5, 4, 7, 6], show the result of the first two iterations, plus the result of initial build-heap to make a max heap, of the Heap Sort algorithm described in the lecture note. Draw the content of the array similar to the exercise solution.

ANSWER:

Initial: [3, 1, 8, 5, 4, 7, 6]
After max-heap: [8, 5, 7, 1, 4, 3, 6]
After 1st iter: [7, 5, 6, 1, 4, 3, 8]
After 2nd iter: [6, 5, 3, 1, 4, 7, 8]

#14. Explain which of the data structures that we have discussed during this quarter would be most useful in each of the following applications. You may choose from the following data structures: stack, queue, list, or priority queue.

a. You are building a system which processes orders that have been placed in an on-line store. The orders should be processed in the order that they were made.

ANSWER: list or (FIFO) queue

b. You are writing a text editor, and wish to implement an "undo" feature, which behaves that way that the Control-z keystroke does in Microsoft Word.

ANSWER: stack

c. You are building a system that maintains a wait-list of passengers on a flight who are waiting for an upgrade to first class. The passenger with the most frequent flier miles is the first to be upgraded when a first class seat becomes available.

ANSWER: priority queue

d. You are building a system that will be used to maintain a list of people who are employed by a company. The system should maintain the list in alphabetical order by the employees' last names, and should support operations to add and remove employees from the list as they are hired by or leave the company.

ANSWER: list

#15. Show the results of inserting 12, 0, 24, 26, and 25 into a hash table of size 13, and a hash function \( H(X) = X \mod 13 \), using the following techniques:

ANSWER:

```
0 | 0 |
---+----+
0---+----+----+
0----+----+----+
0---+----+----+----+
0----+----+----+----+
0---+----+----+----+
---+----+----+----+
---+----+----+----+
---+----+----+----+
---+----+----+----+
```
#16. If we want to minimize the average-case running time of find and insert operations for a very large collection of data, should we store the data in some kind of tree, or in a hash table using quadratic probing? Explain your answer.

**ANSWER:** The worst-case complexity of hash table (of any probing scheme) is $O(n)$, while the worst case complexity for a balanced tree is $O(lgn)$ ...(we never use basic binary search trees in practice, so we assume "some kind of tree" to be a balanced tree.)

So if we are concerned with the worst case, we should use a balanced tree.

#17. Answer the above question for average-case running time.

**ANSWER:** Contrasting to the previous question, we are normally concerned with the average-case -- behaviour which exhibits most often. The average-case complexity of hash table (of quadratic hash table) is $O(1)$, while the average-case complexity of balanced tree is $O(lgn)$. So we should choose hash table.

#18. Given the following graph, show how Dijkstra's algorithm finds the shortest path from A to F.

![Graph Image]

**ANSWER:** Note that a ## indicates positive infinity.

(1) $Q = \{A, B, C, D, E, F, G\}$

\[
\begin{array}{cccccccc}
\text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \text{F} & \text{G} \\
\hline
0 & \## & \## & \## & \## & \## & \## \\
\hline
\end{array}
\]

(2) $Q = \{D, B, C, E, F, G\}$

\[
\begin{array}{cccccccc}
\text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \text{F} & \text{G} \\
\hline
0 & 2 & \## & 1 & \## & \## & \## \\
\hline
\end{array}
\]

(3) $Q = \{B, E, C, F, G\}$
\[ S = \{A, D\} \]
\[ A \quad B \quad C \quad D \quad E \quad F \quad G \]
\[ d = \begin{array}{cccccc}
0 & 2 & \# & 1 & 3 & \# & \# \\
\end{array} \]

(4) \[ Q = \{E, C, F, G\} \]
\[ S = \{A, D, B\} \]
\[ A \quad B \quad C \quad D \quad E \quad F \quad G \]
\[ d = \begin{array}{cccccc}
0 & 2 & 5 & 1 & 3 & \# & \# \\
\end{array} \]

(5) \[ Q = \{C, F, G\} \]
\[ S = \{A, D, B, E\} \]
\[ A \quad B \quad C \quad D \quad E \quad F \quad G \]
\[ d = \begin{array}{cccccc}
0 & 2 & 5 & 1 & 3 & 8 & \# \\
\end{array} \] Note: F and G changed

(6) \[ Q = \{F, G\} \]
\[ S = \{A, D, B, E, C\} \]
\[ A \quad B \quad C \quad D \quad E \quad F \quad G \]
\[ d = \begin{array}{cccccc}
0 & 2 & 5 & 1 & 3 & 7 & 6 \\
\end{array} \] Algorithm terminates at this point because the shortest path to F (from A) is derived.

\[ Q = \{G\} \]
\[ S = \{A, D, B, E, C, F\} \]
\[ A \quad B \quad C \quad D \quad E \quad F \quad G \]
\[ d = \begin{array}{cccccc}
0 & 2 & 5 & 1 & 3 & 7 & 6 \\
\end{array} \]

#19. For the class SCHT class you wrote for HW#6, add a function `listLen` -- This function receives an index (0-based) to the headArray table and returns the length of the linked list coming off the index position (in the headArray table). If the parameter index was invalid (i.e., a negative or larger than the size of the headArray array, the function should return -1.

**ANSWER:**

```cpp
template<typename T>
int SCHT<T>::listLen(int index) const
{
    if (index < 0 || index >= size)
        return -1;
    else {
        int len = 0;
        Node* ptr = headArray[index];
        while (ptr != nullptr) {
            len++;
            ptr = ptr->next;
        }
        return len;
    }
}
```